COSC 465: DISTRIBUTED SYSTEMS ASSIGNMENT

Wilco Milcinovic

EB1/46579/20

1. Mobile and Ubiquitous computing is a new phenomenon in distributed computing discuss.

Ubiquitous Connectivity: Mobile devices, equipped with various communication technologies such as Wi-Fi, Bluetooth, and cellular networks, enable seamless connectivity anytime and anywhere.

Pervasive Computing: The Internet of Things (IoT), creates a pervasive computing environment where interconnected devices collaborate to provide intelligent services.

Context Awareness: Mobile devices are equipped with sensors such as GPS, accelerometers, and gyroscopes, enabling them to sense and understand their users' contexts. By leveraging context-awareness, applications can adapt their behavior based on factors like location, user activity, and environmental conditions, providing personalized and relevant services.

Distributed Data Processing: With the proliferation of mobile devices, computing tasks are distributed across a network of heterogeneous devices. Mobile and ubiquitous computing systems employ distributed data processing techniques to efficiently process data locally on devices, offload computation to cloud servers, or utilize edge computing resources, depending on factors like latency, bandwidth, and energy consumption.

2. Threads usage in distributed systems presents several benefits, explain any two

Improved Performance: Threads allow concurrent execution of multiple tasks within a distributed system. By leveraging parallelism, applications can perform multiple operations simultaneously, thereby reducing overall processing time. For instance, in a distributed web server handling multiple client requests, threads can be used to process incoming requests concurrently, enabling the server to serve more clients simultaneously and reducing response times.

Enhanced Modularity and Maintainability: By encapsulating concurrent tasks within threads, developers can create modular and maintainable code in distributed systems. Each thread can be responsible for a specific task or component, promoting code modularity and separation of concerns. This makes it easier to understand, debug, and maintain the codebase, leading to improved software quality and long-term maintainability.

3. In distributed systems virtualization can take place in many different ways, explain

Operating System-Level Virtualization:

Containers: Containers provide lightweight, OS-level virtualization where applications and their dependencies are packaged together in a container image. Containers share the host OS kernel and resources, but each container is isolated from other containers. Containers are highly portable and efficient, as they do not require a separate OS instance for each container, unlike virtual machines. Technologies such as Docker and Kubernetes have popularized containerization in distributed systems for deploying and managing applications at scale.

Network Virtualization:

Overlay Networks: Overlay networks create virtual networks on top of physical networks, allowing communication between distributed components as if they were part of the same network segment. Technologies like VXLAN and GRE encapsulate network packets within packets, enabling virtual networks to span across physical network boundaries. Overlay networks provide flexibility and scalability in distributed systems, especially in cloud environments where virtual machines or containers may be deployed across multiple physical hosts.

Storage Virtualization:

Virtual Storage Area Networks (SANs): Virtual SANs abstract storage resources from multiple physical storage devices and present them as a single logical storage pool. This allows for centralized management and provisioning of storage resources in distributed systems. Virtual SANs provide features like storage pooling, data replication, and snapshots, enhancing data availability and reliability.

Function Virtualization:

Serverless Computing: Serverless computing abstracts server resources and execution environments, allowing developers to focus on writing functions (also known as serverless functions or FaaS) without managing the underlying infrastructure. Function virtualization dynamically allocates resources to execute functions in response to events or triggers, scaling automatically based on demand. Platforms like AWS Lambda, Google Cloud Functions, and Azure Functions provide serverless computing capabilities in distributed systems.

4. Time synchronization algorithms such as NTP, the time server is passive. Explain a problem that can arise in using a passive time server as opposed to other clock synchronization algorithms

Man-in-the-Middle Attacks: In a man-in-the-middle (MITM) attack, an attacker intercepts communication between the client and the time server and manipulates the time information exchanged between them. The attacker can modify timestamps to skew time measurements, leading to inaccurate time synchronization across the network.

Replay Attacks: In a replay attack, an attacker captures time synchronization messages exchanged between the client and the time server and replays them at a later time. By replaying old messages, the attacker can trick the client into believing that it is synchronizing with the legitimate time server when, in fact, the time information is outdated or manipulated.

Spoofing Attacks: In a spoofing attack, an attacker impersonates the legitimate time server and provides false time information to the client, leading to incorrect time synchronization across the network.

5. with use of appropriate examples and sketches, discuss mutual exclusion and how it is achieved in distributed systems

Centralized Approaches:

In centralized mutual exclusion algorithms, a single node or process acts as a coordinator responsible for granting access to the shared resource. Examples include the centralized approach using a token-based algorithm or a centralized server-based algorithm.

Token-based Algorithm: In this approach, a special token is passed among processes, and only the process holding the token can access the critical section. When a process completes its operation in the critical section, it passes the token to another process. This ensures that only one process has access to the critical section at any given time. Below is a sketch illustrating how the token is passed among processes:

Distributed Approaches:

Distributed mutual exclusion algorithms distribute the responsibility of coordinating access to the critical section among multiple nodes in the system. Examples include the Ricart-Agrawala algorithm, the Maekawa algorithm, and the Lamport's Bakery algorithm.

Maekawa Algorithm: This algorithm partitions processes into disjoint sets called "quorums." Each process maintains a local request queue and sends requests only to processes in its quorum. Processes grant permission to enter the critical section based on a voting mechanism within their quorums. Below is a sketch illustrating the quorum structure and communication among processes in the Maekawa algorithm:

6. Discuss scalability, types and importance

Scalability is the ability of a system to handle increasing workloads or growing demands while maintaining or improving its performance, capacity, and efficiency.

Types:

Vertical Scalability (Scaling Up):

Vertical scalability involves increasing the capacity of individual components within the system, typically by adding more resources to existing nodes. For example, upgrading a server's CPU, memory, or storage capacity to handle increased load.

Horizontal Scalability (Scaling Out):

Horizontal scalability involves adding more nodes or instances to the system to distribute the workload across multiple machines. This approach enables the system to handle increased load by parallelizing tasks across multiple nodes. Horizontal scalability is more flexible and can potentially handle larger workloads than vertical scalability.

Importance:

Handling Increased Workloads:

Scalability ensures that a distributed system can accommodate growing workloads, whether due to increased user traffic, data volume, or computational demands.

Improved Performance and Responsiveness:

Scalable distributed systems can distribute workloads efficiently across multiple nodes, reducing the risk of bottlenecks and ensuring optimal resource utilization.

Flexibility and Adaptability:

Scalability provides flexibility to adapt to changing requirements and evolving business needs. Distributed systems that can scale easily can respond to fluctuations in demand, seasonal variations, or unexpected spikes in workload without service degradation or downtime.

Cost-Effectiveness:

Scalability enables distributed systems to scale resources up or down based on demand, optimizing resource utilization and minimizing operational costs. By scaling horizontally and leveraging cloud computing resources, organizations can avoid over-provisioning and only pay for the resources they consume.

7. Discuss heterogeneity and its purpose in mobile code

Heterogeneity refers to the diversity or variation in characteristics among different components or entities within a system.

Platform Independence:

One of the primary purposes of mobile code is to enable platform-independent execution of applications across diverse computing environments. Heterogeneity allows mobile code to run seamlessly on different hardware architectures, operating systems, and software platforms without requiring modifications or recompilation. This flexibility is particularly valuable in distributed systems where the underlying infrastructure may vary significantly.

Dynamic Adaptation:

Heterogeneity enables mobile code to dynamically adapt to changing runtime environments and system configurations. Mobile code systems can detect and utilize available resources, such as CPU capabilities, memory size, network bandwidth, and peripheral devices, to optimize application performance and resource utilization.

Load Balancing and Resource Management:

Heterogeneity facilitates load balancing and resource management in distributed systems by enabling the distribution of computational tasks across diverse computing nodes. Mobile code systems can dynamically allocate tasks to nodes based on their capabilities, workload, and proximity to data sources, ensuring optimal resource utilization and performance scalability.

Fault Tolerance and Resilience:

Heterogeneity enhances fault tolerance and resilience in distributed systems by diversifying the execution environment and reducing single points of failure. Mobile code systems can replicate and distribute critical components across heterogeneous nodes, mitigating the impact of hardware failures, software errors, or network disruptions.

Interoperability and Integration:

Heterogeneity fosters interoperability and integration among disparate systems, protocols, and technologies in distributed environments. Mobile code systems can bridge the gap between different platforms, enabling seamless communication, data exchange, and collaboration across heterogeneous devices, networks, and services.

Scalability and Performance Optimization:

Heterogeneity supports scalability and performance optimization in distributed systems by leveraging the diversity of computing resources and architectures. Mobile code systems can exploit parallelism, concurrency, and distributed processing techniques to maximize throughput, reduce latency, and improve responsiveness across heterogeneous nodes.